Raissa Nunes Peret (2024.1.08.021) e Ana Flávia Freiria Rodrigues (2024.1.08.001)

Lista de Exercícios de Engenharia de Software – Implementação e Testes

---

1. Uma das atividades do fluxo de implementação é a construção de protótipos. Qual o objetivo dessa prática? Em que momento (fases: concepção, elaboração, construção e transição) de um processo tradicional iterativo incremental a prototipação pode ser aplicada?

   A prototipação tem como objetivo criar uma versão inicial de um sistema para visualizar e testar suas funcionalidades antes de desenvolver a versão final. Isso ajuda a esclarecer os requisitos, identificar problemas e melhorar a comunicação com os stakeholders. A prototipação pode ser aplicada em diversas fases de um processo iterativo incremental, mas é mais comum nas fases de *concepção e elaboração*, pois é nesse momento que as ideias e funcionalidades estão sendo definidas e validadas.

---

2. O que são boas práticas de codificação? Cite três exemplos e explique sua importância.

   Boas práticas de codificação são técnicas e convenções que ajudam a escrever um código mais claro, eficiente e fácil de manter. Três exemplos importantes são:

- *Nomeação clara de variáveis e funções:* Nomes descritivos tornam o código mais compreensível e ajudam outros desenvolvedores (ou até o próprio autor, no futuro) a entender rapidamente o que o código faz.
- *Documentação do código:* Comentários e documentação são essenciais para explicar por que algo foi feito de determinada maneira, o que facilita a manutenção e futuras modificações.
- *Uso correto de indentação*: Um código bem indentado melhora a legibilidade e evita erros, principalmente em linguagens onde a indentação define o bloco de execução.

---

3. O que é refatoração de código? Quais são os benefícios e quando deve ser aplicada?

   Refatoração de código é o processo de reestruturar o código existente sem alterar seu comportamento externo. O objetivo é melhorar a legibilidade, reduzir a complexidade e facilitar futuras manutenções. Refatorar o código traz benefícios como:

- *Melhoria na legibilidade*: Código mais limpo e organizado é mais fácil de entender.
- *Facilidade de manutenção*: Código bem estruturado é mais fácil de corrigir e expandir.

- *Redução de dívidas técnicas:* Erros e soluções temporárias podem ser eliminados durante a refatoração.

  Deve ser aplicada quando o código começa a ficar difícil de entender ou modificar, ou como parte do desenvolvimento contínuo, em pequenos incrementos, para manter o código saudável.

---

4. Como as ferramentas de controle de versão, como Git, ajudam na construção de software? Dê exemplos de práticas recomendadas.

   Ferramentas de controle de versão, como o Git, ajudam na construção de software ao permitir que várias pessoas trabalhem no mesmo projeto simultaneamente, mantendo um histórico das mudanças feitas. Elas facilitam a colaboração e evitam que o trabalho de um desenvolvedor sobreponha o de outro. Práticas recomendadas no uso dessas ferramentas incluem:

- *Commits frequentes e significativos:* Fazer commits com mensagens claras facilita entender o histórico do projeto e reverter alterações, se necessário.
- *Branching:* Utilizar branches (ramificações) para separar o desenvolvimento de novas funcionalidades ou correções de bugs, mantendo o código principal (geralmente a branch "main") sempre estável.
- *Pull requests e code reviews*: Usar pull requests e revisões de código antes de mesclar alterações na branch principal garante que o código seja revisado por outras pessoas e que erros sejam identificados antes de chegar à produção.

---

5. Explique, concordando ou discordando, a seguinte frase: “A definição e execução de testes são tarefas exclusivas do fluxo de teste”.

   Discordo dessa afirmação. Embora o fluxo de testes seja responsável pela execução e definição dos testes formais, testar o software não é uma responsabilidade exclusiva dessa fase. O processo de testes deve estar presente em todas as etapas do desenvolvimento. Durante a *fase de implementação*, por exemplo, desenvolvedores podem e devem realizar testes automatizados, como testes unitários, para garantir que as partes do código funcionem como esperado. Além disso, na fase de *concepção e elaboração*, a criação de cenários de teste pode ajudar a definir requisitos mais claros e a planejar soluções melhores. Assim, a qualidade do software é uma responsabilidade compartilhada entre todas as fases do projeto.

---

6. Descreva a importância da documentação na fase de construção do software. Que tipos de documentação você considera essenciais?

   A documentação é fundamental durante a fase de construção, pois serve como referência tanto para a equipe de desenvolvimento quanto para futuros mantenedores do sistema. Ela ajuda a garantir que todos estejam alinhados e

compreendam como as funcionalidades estão sendo implementadas. Tipos de documentação essenciais:

- *Documentação de requisitos*: Define claramente o que o software deve fazer, garantindo que as necessidades do cliente sejam atendidas.
- *Documentação de código:* Comentários e explicações no próprio código ajudam outros desenvolvedores (ou o autor original, no futuro) a entender o funcionamento de trechos complexos.
- *Manuais de usuário e de instalação:* Cruciais para garantir que o sistema possa ser usado e configurado corretamente, tanto pelos usuários finais quanto pela equipe técnica.

---

7. Qual a diferença entre atividades de inspeção de implementação e teste?

   Inspeção de implementação é uma atividade mais focada na revisão do código e do design. O objetivo é encontrar problemas relacionados à qualidade do código, como má estruturação, violações de padrões de design e oportunidades de refatoração. Essa inspeção é geralmente feita por desenvolvedores e envolve revisões manuais ou automatizadas. Já os testes envolvem a execução do software para garantir que ele esteja funcionando conforme esperado. O foco dos testes é identificar problemas de comportamento ou falhas na funcionalidade, simulando cenários reais de uso. Enquanto a inspeção analisa o código, os testes validam o comportamento do software.

---

8. Explique a seguinte frase, concordando ou discordando: “O objetivo das metodologias de teste é maximizar a cobertura”.

   Concordo, maximizar a cobertura é sim um dos objetivos das metodologias de teste, mas não é o único, nem sempre o mais importante. *Cobertura de teste* se refere a quanto do código ou das funcionalidades do sistema é verificado pelos testes. No entanto, simplesmente maximizar a cobertura não garante que o software estará livre de bugs ou que todas as situações reais de uso foram contempladas.

---

9. Explique a seguinte frase, concordando ou discordando: “Assim como as atividades de desenho, as atividades de teste devem ser feitas por especialistas”.

   Concordo parcialmente, as atividades de teste podem ser feitas por especialistas, especialmente em projetos mais complexos, onde o conhecimento profundo das técnicas de teste e da estrutura do sistema é essencial para identificar falhas difíceis de encontrar. No entanto, testes mais simples, como testes unitários e testes funcionais básicos, podem e devem ser realizados por desenvolvedores ou até mesmo por outros membros da equipe de desenvolvimento. Além disso, testes manuais que simulam o uso do sistema muitas vezes podem ser realizados por usuários finais ou membros da equipe que não sejam especialistas técnicos, pois trazem uma perspectiva prática e mais próxima do uso real.

---

10. Dê um exemplo, no contexto de um sistema qualquer, de teste que utilize o método caixa branca e outro que utilize o método caixa preta.

    Teste caixa branca: Nesse tipo de teste, o testador tem acesso ao código e se preocupa em testar fluxos específicos e a lógica interna do software. Um exemplo seria verificar se uma função de *cálculo de juros compostos* está percorrendo corretamente os loops e somando os valores de forma adequada. O desenvolvedor sabe como o algoritmo funciona e testa as estruturas internas.

    Teste caixa preta: Aqui, o testador não tem conhecimento do código e foca no comportamento do sistema, baseado nos requisitos. Um exemplo seria testar uma *tela de login* de um sistema. O testador inseriria diferentes combinações de usuário e senha para ver se o sistema aceita as credenciais corretas e rejeita as erradas, sem se preocupar com como o código por trás está implementado.

---

11. Existem diferentes categorias de teste. Qual a diferença entre um teste de aceitação e um teste de unidade? Dê um exemplo de teste de sistema e outro de integração. Para que servem os testes de regressão?

- *Teste de unidade*: Testa partes individuais do código, como funções ou métodos. Ele é feito isoladamente para garantir que cada unidade de código funciona corretamente. Por exemplo, testar uma função que soma dois números e retorna o resultado.
- *Teste de aceitação*: Este é um teste mais amplo, feito para verificar se o sistema atende aos requisitos do cliente e pode ser entregue. Ele é feito em um ambiente o mais próximo possível do real. Um exemplo seria um teste completo de um sistema de gestão de pedidos, verificando se todas as funcionalidades (fazer pedidos, gerar relatórios, etc.) estão funcionando conforme o solicitado pelo cliente.

Exemplo de teste de sistema e teste de integração:

- Teste de sistema: Verifica o sistema completo, avaliando se todas as funcionalidades estão funcionando como esperado. Por exemplo, testar se um site de e-commerce permite que o usuário navegue pelos produtos, adicione itens ao carrinho, finalize a compra e receba uma confirmação.
- Teste de integração: Foca na comunicação entre diferentes partes do sistema. Um exemplo seria testar se o sistema de login (módulo A) está se comunicando corretamente com o sistema de gerenciamento de usuários (módulo B) e permitindo que usuários autenticados acessem a área restrita.

Testes de regressão: São utilizados para garantir que, após uma modificação no código (como a adição de uma nova funcionalidade ou correção de um bug), as funcionalidades que já estavam funcionando corretamente antes não foram afetadas. Eles ajudam a prevenir o surgimento de novos erros em áreas que já haviam sido testadas.

---

12. O que é um caso de teste? Dê um exemplo simples de um caso de teste de um sistema de informação típico.

    Um caso de teste é um conjunto de condições e variáveis que definem como o teste será executado para verificar se uma funcionalidade do sistema está funcionando corretamente. Ele descreve o que deve ser feito, os dados de entrada, as ações esperadas e o resultado esperado. Ex:

Sistema: **Sistema de gerenciamento de biblioteca**
Funcionalidade: **Empréstimo de livros**

- **Objetivo**: Verificar se um livro pode ser emprestado corretamente.
- **Cenário**: O usuário seleciona um livro disponível e solicita o empréstimo.
- **Dados de entrada**: ID do usuário (12345), ID do livro (6789).
- **Passos**:
    1. O usuário faz login no sistema.
    2. O usuário pesquisa pelo livro pelo ID.
    3. O sistema exibe a disponibilidade do livro.
    4. O usuário solicita o empréstimo.
    5. O sistema confirma o empréstimo e exibe a data de devolução.
- **Resultado esperado**: O sistema deve registrar o empréstimo com sucesso e exibir a data correta de devolução.

---

13. Explique a seguinte frase, concordando ou discordando: “Uma vez que testes são atividades exaustivas, não tem como saber em que momento é melhor parar de testar”.

    Discordo, embora testar software possa parecer uma atividade interminável, com muitos cenários possíveis a serem cobertos, há maneiras de saber quando parar de testar. O objetivo dos testes não é cobrir absolutamente todas as combinações de situações, mas sim atingir uma boa confiança na qualidade do sistema. Em geral, os testes podem ser considerados suficientes quando: Os requisitos foram completamente testados, nenhum bug crítico foi encontrado nas últimas execuções de teste, a cobertura de código é satisfatória.

    Saber quando parar também depende de fatores práticos, como prazos e recursos disponíveis. Parar de testar pode ser uma decisão estratégica, equilibrando o risco de falhas com o tempo e esforço envolvidos em continuar os testes.

---

14. Como a técnica de TDD (Test-Driven Development) pode influenciar a qualidade do software produzido? Descreva o processo.

O TDD (Desenvolvimento Orientado a Testes) é uma técnica que pode melhorar bastante a qualidade do software. No TDD, o processo é invertido: antes de escrever o código, o desenvolvedor escreve *um teste para a funcionalidade* desejada. Somente depois de ter esse teste, ele cria o código que fará o teste passar. O ciclo segue este padrão:

1. *Escreva um teste: Crie um teste para a nova funcionalidade ou para corrigir um erro específico.*
2. *Escreva o código: Apenas o suficiente para fazer o teste passar.*
3. *Refatore o código: Após garantir que o teste passa, melhore a qualidade do código sem alterar sua funcionalidade.*

Esse processo ajuda a garantir que o código foi desenvolvido para atender a um propósito claro e testável desde o início, o que melhora a confiabilidade e a qualidade geral do software. Ao focar nos testes desde o começo, o desenvolvedor cria um código mais limpo.

---

15. Quais são os principais desafios que podem ser enfrentados durante a fase de testes de um software? Como superá-lo?

- Cobertura insuficiente de testes: Cobrir todas as funcionalidades e caminhos de execução do software pode ser difícil, especialmente em sistemas complexos. Para superar isso, é importante priorizar os testes, focando primeiro nas áreas mais críticas, como funcionalidades essenciais e partes do sistema mais propensas a falhas.
- Ambientes de teste inconsistentes: Muitas vezes, o ambiente de testes não reflete o ambiente real onde o software será executado, o que pode levar a bugs não detectados. Para superar isso, é importante ter ambientes de testes que simulem o ambiente de produção o mais fielmente possível, inclusive replicando a carga e o uso real.
- Gestão do tempo e prazos apertados: O teste muitas vezes é pressionado por prazos, o que pode resultar em menos tempo para a execução completa dos testes. Para lidar com esse desafio, técnicas como testes automatizados e priorização de testes (testando primeiro as funcionalidades de maior risco) podem ser uma solução eficaz.
- Falta de ferramentas adequadas: Sem as ferramentas certas, testar pode ser um processo manual e propenso a erros. Investir em ferramentas de automação de testes e controle de versão, como o Git, pode melhorar a eficiência e a eficácia dos testes.